МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут автоматики, кібернетики та обчислювальної техніки

Кафедра комп'ютерних наук та прикладної математики

«До захисту допущена»

Завідувач кафедри комп'ютерних наук

та прикладної математики

_____ Турбал Ю.В.

«_____» _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка вебзастосунку для вантажних перевезень»

Виконав: Редзель Юрій Сергійович _____

(прізвище, ім'я, по батькові) (підпис)

група КН-41

Керівник: старший викладач Харів Н.О. _____

(науковий ступінь, вчене звання, посада, прізвище та ініціали) (підпис)

Рівне – 2023

Національний університет водного господарства та природокористування

Навчально-науковий інститут автоматики, кібернетики та обчислювальної техніки
Кафедра **комп'ютерних наук та прикладної математики**
Рівень вищої освіти **бакалавр**
Галузь знань **12 «Інформаційні технології»**
Спеціальність **122 «Комп'ютерні науки»**

«ЗАТВЕРДЖУЮ»
Завідувач кафедри
комп'ютерних наук та прикладної математики
д.т.н., професор Турбал Ю.В.

«____» _____2023 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Редзелю Юрію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Розробка вебзастосунку для вантажних перевезень*
керівник роботи *Харів Наталія Олексіївна, старший викладач кафедри комп'ютерних наук та прикладної математики*
затверджені наказом вищого навчального закладу від «*19*» *квітня 2023 року С №-449*
2. Термін здачі студентом закінченої роботи *29.05.2023*
3. Вихідні дані до роботи*: документація замовника*
4. Зміст розрахунково-пояснювальної записки *Розділ 1. Огляд предметної області. Розділ 2. Інструменти розробки. Розділ 3. Етапи розробки. Розділ 4. Програмна реалізація. Висновок*
5. Перелік графічного матеріалу *мультимедійна презентація*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1	*Харів Н.О., ст. викладач*		
Розділ 2	*Харів Н.О., ст. викладач*		
Розділ 3	*Харів Н.О., ст. викладач*		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	*Опрацювання літератури та інтернет-джерел на задану тему.*		
2	*Узгодження із замовником.*		
3	*Опрацювання завдання та виявлення ключових аспектів*		
4	*Вибір технологій для реалізації серверної частини*		
5	*Вибір технологій для реалізації клієнтської частини*		
6	*Аналіз та вивчення документації для реалізації*		
7	*Програмна реалізація.*		
8	*Тестування та виправлення помилок.*		
9	*Оформлення теоретичної частини*		

Студент _____ (Редзель Ю.С)

Керівник кваліфікаційної роботи _____ (Харів Н.О)

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота: 43 листи, 16 малюнків, 4 додатки

Мета: Розробка вебзастосунку для вантажних перевезень Україною.

Завдання: Розробити вебзастосунок для вантажних перевезень Україною, який буде зручний для використання клієнтом та легкий у навчанні працівників

Для досягнення поставленої мети, потрібно виконати наступні пункти:

- Проаналізувати вебзастосунки у галузі перевезень та визначити їх проблеми
- Проаналізувати та визначити проблеми, які виникають під час співпраці клієнта та працівника
- Вивчити проблеми, з якими стикаються працівники при роботі з клієнтом
- Створити план реалізації додатку (визначення технологій та функціоналу)
- Розробка функціоналу
- Розробка інтуїтивно зрозумілого та привабливого користувацького інтерфейсу
- Розробка інтерфейсу, який направлений на швидке навчання персоналу
- Впровадження та підтримка

Результати: Розроблений вебзастосунок для вантажних перевезень Україною, який надає зручний та ефективний користувальницький інтерфейс, швидкий у освоєнні та функціонально потужний застосунок для працівника.

Ключові слова: вебзастосунок, серверна та клієнтська частина, інтерфейс користувача, користувач, база даних, працівник.

АНОТАЦІЯ

У кваліфікаційній роботі розглянуто розробку вебзастосунку для транспортних перевезень Україною. Головною метою є реалізація зручного, ефективного та естетично красивого вебзастосунку для користувачів, у яких є потреба скористатися послугами транспортного перевезення, та швидкого у розумінні та зручного у використанні вебзастосунку для працівників.

У роботі використовується React JS, один із найзручніших у роботі з кодом фреймворк, який полегшує написання клієнтської частини, та додаткові бібліотеки, які призначені для роботи з сервером та реалізації ефективного та естетичного клієнтського інтерфейсу. За основу реалізації серверної частини взято Node JS, який у поєднанні з бібліотеками та фреймворками надає необхідні інструменти для реалізації потужної серверної частини, яка швидко та зручно взаємодіє з клієнтською частиною та має відповідний інструментарій для збереження інформації в базу даних, яку потребує вебзастосунок.

Вебзастосунок забезпечує користувача функціональним інтерфейсом, в якому можливо слідкувати за статусом своїх замовлень, вести комунікацію з персоналом та редагувати потрібну користувачу інформацію. А для працівника надається швидкий в освоєнні та зручний у використанні інструмент для взаємодії з користувачем та його замовленнями.

У результаті роботи був створений зручний, ефективний та естетично красивий вебзастосунок, який вирішує достатню кількість проблем у сфері вантажних перевезень, або у сфері надання послуг таких як:

- Комунікація між працівником та клієнтом
- Проблема якості обслуговування
- Проблема заощадження часу клієнта
- Проблема навчання персоналу необхідним для ефективного обслуговування інструментарієм

ВСТУП

Світовий процес переходу суспільства до інформаційно-комп'ютерних технологій має революційний вплив на всі сфери суспільного життя і вимагає суттєвих змін у багатьох галузях діяльності людини. Раніше для пошуку інформації, розваг чи реалізації наших потреб, потрібно було затратити немалу кількість сил та часу. Але з приходом нових можливостей у світ комп'ютерних технологій усі ці потреби можливо задовольнити, не виходячи з дому. Саме всесвітня мережа-інтернет та вебзастосунки для неї допомагають нам у вирішенні будь-яких завдань, у побутових проблемах, навчальному процесі, дозвіллі або ж проблемах, пов'язаних з роботою.

Однією з таких проблем є досить актуальна на цей час – проблема комунікації клієнта з працівником та заощадження їхнього часу. Ця проблема стосується усіх галузей діяльності, які надають послуги, пов'язані з купівлею або замовленням. Тому галузь транспортних перевезень, а саме перевезення своїх особистих речей, або речей, які потрібно доставити з точки А в точку В підпадає під цю категорію. Досить не просто створити зручні умови, хорошу підтримку та злагоджену роботу з клієнтом у галузі транспортування, щоб він залишився задоволеним і звернувся ще раз. Тому великі компанії, у яких достатня кількість замовників, завжди в пошуках шляхів для автоматизації та полегшення умов надання своїх послуг і, як згадувалося раніше, одними з ключових шляхів подолання таких труднощів виступають вебзастосунки.

Актуальні проблеми, з якими можна стикнутися у сфері надання послуг:

- **Недостатня комунікація** – клієнти можуть стикатися з проблемою недостатньої комунікації з постачальником послуг. Це може включати відсутність відповіді на запити, незрозумілість інформації або погану співпрацю. Недостатня комунікація може призводити до незадоволеності клієнтів та втрати довіри.

- **Швидкість та якість обслуговування** – якість обслуговування є критично важливою у сфері надання послуг. Непрофесійне обслуговування, таке

як нешвидка реакція на запити клієнтів, некомпетентність працівників або незадовільне виконання обіцянок, може призвести до негативного досвіду клієнтів та втрати бізнесу. Тому важливо створити інструментарій, за допомогою якого працівник зможе швидко та без труднощів реагувати на запити клієнта

- **Проблеми з управлінням часом** – недостатня організація та управління часом можуть призвести до затримок у наданні послуг. Це може бути спричинено неоптимізованим середовищем для надання послуг. Важкий у розумінні працівника інструментарій та неефективно-функціональні шляхи взаємодії працівника з клієнтом та його замовленнями.

Тому все більше компаній, які націлені на довгострокову співпрацю з клієнтами, полегшення життєвого процесу роботи з клієнтом та автоматизацію бізнес-процесів, схильні до думки, що вебзастосунки відіграють одну з ключових ролей у життєдіяльності сфери надання послуг. Створення вебзастосунку, в якому клієнт може легко, швидко та зручно маніпулювати своїм замовлення, прослідковувати за статусом його виконання та у випадку виникнення запитань створювати запити на спілкування з оператором, а працівник, в першу чергу зміг швидко та легко звикнути до використання цих технологій – навчання персоналу, постає досить важливою задачею, або ж складною проблемою, яку можна успішно вирішити.

РОЗДІЛ 1
ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Загальні відомості про вебзастосунки транспортних перевезень

Вебзастосунки у галузі транспортних перевезень відіграють важливу роль у роботі з клієнтом. Вони надають зручний функціонал та заощаджують час клієнта та компанії, яка направлена на довгострокову та ефективну співпрацю з клієнтами. Створення ефективного та водночас естетично красивого вебзастосунку надає компанії широке коло клієнтської бази, адже вебзастосунок надає не тільки інструменти надання послуг та вирішення проблемних питань з клієнтом, але й приваблення клієнта.

Вебзастосунки відіграють вирішальну роль у полегшенні комунікації та співпраці з клієнтами. За допомогою, яких можна впровадити, такі заходи як:

- Онлайн-чат та системи обміну повідомленнями: Вебзастосунки можуть включати в себе вбудовані системи онлайн-чату або обміну повідомленнями, які дозволяють клієнтам спілкуватися з представниками компанії в режимі реального часу. Це дає можливість швидко відповідати на запитання, надавати підтримку та вирішувати проблеми без необхідності чекати на відповідь по електронній пошті або телефону.
- Онлайн-форми та системи керування запитами: Вебзастосунки можуть включати форми запитів або системи керування запитами, які дозволяють клієнтам зручно подавати свої запити, проблеми або скарги. Це спрощує процес спілкування, дозволяючи клієнтам точно вказувати свої потреби та отримувати швидкі та цілеспрямовані відповіді.
- Віртуальні панелі керування клієнта: Вебзастосунки можуть мати віртуальні панелі керування, доступ до яких мають клієнти. Ці панелі надають клієнтам зручний спосіб керування своїми обліковими записами, персональною інформацією, замовленнями та історією взаємодії з компанією. Клієнти можуть виконувати різні дії, такі як зміна адреси доставки, відстеження.

На сьогодення припадає достатня кількість вебзастосунків у галузі транспортних перевезень, але не у всіх забезпечений інструментарій для вирішення проблем користувачів – клієнтів. В деяких, навіть, якщо функціональна база вебзастосунку зроблена на 100%, функціонал надає повний спектр вирішення проблематики клієнта, можливо недостатньо проаналізовано зручність використання та реалізації інтерфейсу користувача. Інтерфейс користувача, при реалізації вебзастосунку, який спрямований насамперед для надання послуг, реалізація зручного та водночас повністю зрозумілого інтерфейсу користувача посідає одну з ключових проблем, з якими можна стикнутися при створенні вебпродукту.

Для забезпечення задоволення користувачів вебзастосунку важливо створити інтуїтивно зрозумілий та зручний інтерфейс, який дозволяє легко здійснювати потрібні дії та досягати поставлених цілей. Вебзастосунки повинні бути придатними для використання на різних пристроях, таких як комп'ютери, смартфони, планшети, забезпечуючи адаптивний дизайн та респонсивність.

Тому при реалізації, насамперед потрібно проаналізувати вебзастосунки, які знаходяться на використанні у цій галузі, провести аналіз клієнтів, з якими у майбутньому працівникам доведеться працювати, та знайти інструментарій, який буде спрямований на вирішення проблем зручності та функціональності надання послуг.

Для початку необхідно визначити ключові поняття, які будуть спрямовані на полегшення розуміння процесу розробки вебзастосунку.

1.2. Ключові поняття

Клієнт транспортних перевезень – це особа або організація, яка замовляє послуги перевезення вантажу або пасажирів від одного місця до іншого. Клієнти транспортних перевезень можуть бути приватними особами, бізнесами, логістичними компаніями, вантажними агентами, туристичними агентствами тощо.

Користувач вебзастосунку – це особа або організація, яка використовує вебзастосунок для досягнення певних цілей або виконання певних завдань.

Користувач може бути будь-яким відвідувачем вебсайту, який використовує деякі функціональні можливості, або зареєстрованим користувачем з особистим обліковим записом.

Працівники – особи, які відповідають за надання технічної підтримки користувачам, вирішення їхніх проблем та відповідь на запити.

Як правило, вони повинні вміло та ефективно надавати такі послуги як:

- Прийом замовлень
- Обробка замовлень
- Управління базою даних
- Вирішення технічних проблем
- Комунікація з користувачами
- Супровід клієнта
- Підтримка користувача

Замовлення на транспортне перевезення вантажу через вебзастосунок є процесом організації та бронювання транспорту для переміщення вантажу з одного місця в інше.

Замовлення на транспортне перевезення вантажу зазвичай включає в себе ряд етапів. Спочатку клієнт вводить відповідну інформацію про вантаж, таку як його характеристики, вагу, розміри, точку походження та місце призначення

Комунікація між працівником та клієнтом, користувачем є процесом обміну інформацією, ідеями та сприйняттями з метою досягнення спільних цілей. Це взаємодія, яка відбувається з метою зрозуміти потреби та очікування клієнта або користувача, відповісти на їх запити та надати необхідну допомогу.

Відгук клієнта є унікальною думкою, яку надає клієнт після отримання послуги чи використання продукту або сервісу. Це вираження думки, вражень та задоволення клієнта щодо свого досвіду. За рахунок відгуків користувачі, які ще не стали клієнтами, можуть переконатися у ефективності та хорошій роботі представників компанії і самої компанії.

Кабінет клієнта – це захищений обліковий запис, до якого клієнт має доступ після реєстрації або авторизації в системі. Цей кабінет надає клієнту

можливість взаємодіяти з вебзастосунком, керувати своїм обліковим записом та отримувати персоналізовану інформацію та послуги.

Кабінет працівника – це кабінет надає працівнику доступ до всієї необхідної інформації та функцій, пов’язаних з обробкою замовлень і спілкуванням з користувачами. У ньому працівник може переглядати та керувати замовленнями, отримувати повідомлення від користувачів, вносити зміни до даних замовлень та надавати відповіді на запити клієнтів.

Редагування замовлення як працівником, так і користувачем є процесом зміни параметрів або деталей замовлення з метою внесення необхідних змін або корекції інформації. Це дозволяє забезпечити гнучкість та адаптованість замовлення під потреби та вимоги користувача.

Чат між користувачем та працівником – це інтерактивний засіб спілкування, який дозволяє користувачам взаємодіяти з працівниками в реальному часі. Це форма електронного спілкування, в якій повідомлення передаються швидко та миттєво, надаючи можливість обміну інформацією безпосередньо через текстові повідомлення.

1.3. Формулювання задачі

Метою кваліфікаційної роботи є розробка функціонального та зручного для використання вебзастосунку для вантажних перевезень Україною, з одного боку легкого для освоєння працівнику та потужного і зручного у використанні для клієнта, користувача.

Розроблена програмна система надаватиме функціонал для:

Клієнта:

- Інтерфейс
- Реєстрація та особистий кабінет
- Редагування особистих даних
- Калькулятор ціни замовлення
- Створення, редагування та оплата замовлення
- Оповіщення про стан та іншу інформацію, про замовлення на пошту
- Створення запиту на дзвінок

- Створення запиту на чат, для спілкування з працівником

Працівника:

- Зручний та швидкий в освоєнні інтерфейс застосунку
- Перегляд та редагування усіх замовлень клієнтів
- Перегляд та редагування особистих даних клієнта
- Перегляд та редагування запитів на дзвінок
- Створення замовлення, через працівника
- Перегляд та взаємодія з чатами клієнтів

РОЗДІЛ 2
ІНСТРУМЕНТИ РОЗРОБКИ

2.1. Вступ

Для початку реалізації вебзастосунку, потрібно чітко розуміти, що є серверна частина та клієнтська частина додатку. Серверна частина вебзастосунку – це програмне забезпечення, яке виконується на сервері і забезпечує обробку запитів вебклієнтів, збереження та керування даними, логіку бізнес-процесів та відповідь на запити. Клієнтська частина вебзастосунку – це програмний код і графічний інтерфейс, що реалізовані на боці користувача і відповідають за взаємодію з користувачем та відображення вебзастосунку на його пристрої.

Клієнтська частина вебзастосунку може бути розроблена з використанням різних технологій, таких як HTML, CSS і JavaScript, а також фреймворків і бібліотек для створення більш складного функціоналу та поліпшення користувацького досвіду.

Клієнтська частина може включати такі елементи:

Графічний інтерфейс, логіка взаємодії та взаємодія інтерфейсу з базою даних

Серверна частина вебзастосунку може включати різні компоненти, такі як вебсервер, база даних, мікросервіси, додаткові бібліотеки та фреймворки, а також спеціалізовані модулі для виконання конкретних завдань. Вона виконує обробку запитів, перевірку даних, доступ до бази даних, взаємодію з іншими системами та інші функції, необхідні для правильної роботи вебзастосунку.

Основні завдання серверної частини вебзастосунку включають:

- Прийом та обробку запитів від вебклієнтів: Серверна частина отримує HTTP-запити від клієнтів, розбирає їх та виконує необхідні дії відповідно до логіки застосунку.
- Робота з базою даних: Вебзастосунки часто потребують зберігання та управління даними. Серверна частина взаємодіє з базою даних, виконує запити до неї, отримує та зберігає дані.

Загалом, серверна частина вебзастосунку є ключовим елементом, який забезпечує обробку запитів користувачів, збереження даних та виконання логіки бізнес-процесів, і вона має бути розроблена з урахуванням унікальних потреб та вимог конкретного вебзастосунку.

2.2. Серверна частина

Node JS – це середовище виконання JavaScript, яке взято за основу створення серверної частини, яке дозволяє розробникам виконувати JavaScript-код на серверній стороні. Воно базується на движку V8, який також використовується в браузері Google Chrome, і надає можливість виконувати JavaScript-код поза контекстом веббраузера. Node.js має активну та розширену екосистему, яка включає тисячі модулів та пакетів, доступних через менеджер пакетів npm.

Node.js надає можливість розробляти серверні додатки з використанням JavaScript, що дає розробникам можливість використовувати їхні знання та навички в одній мові на обох сторонах – клієнтській та серверній.

Швидкодія та масштабованість: Node.js славиться своєю високою продуктивністю завдяки використанню потужного движка V8. Воно може обробляти велику кількість одночасних підключень з мінімальними накладними витратами, що робить його дуже масштабованим для створення високонавантажених додатків.

У більшості застосунків прийнято за основу працювати з Node JS у парі з Express JS для швидкої та зручної взаємодії з запитами на сервер.

Express.js – це легковаговий фреймворк для розробки вебдодатків на основі Node.js. Він надає простий та ефективний спосіб створення вебсерверів і API. Основна мета Express.js - спростити розробку вебдодатків, забезпечуючи мінімалістичний набір функцій та гнучку архітектуру. Express.js є дуже популярним фреймворком серед розробників Node.js завдяки своїй простоті, гнучкості і розширюваності

MongoDB база даних – це документо-орієнтована система керування базами даних (СКБД), яка використовує JSON-подібні документи для зберігання

та організації даних. Вона входить до сімейства NoSQL баз даних і використовується для зберігання та обробки великого обсягу структурованих та неструктурованих даних.

MongoDB використовує документи BSON (Binary JSON), що дозволяє зберігати дані у вигляді структурованих документів. Кожен документ може мати свою унікальну структуру, що дозволяє гнучко зберігати дані без необхідності попередньої дефініції схеми

Socket.IO – це бібліотека JavaScript, яка дозволяє реалізувати двосторонній зв'язок між вебпереглядачем (клієнтом) і сервером за допомогою технології WebSockets. Вона забезпечує можливість взаємодії в реальному часі, передачу даних та подій між клієнтом і сервером в обох напрямках.

Основні особливості та можливості Socket.IO:

- Реальний час: Socket.IO дозволяє надсилати дані між клієнтом і сервером у реальному часі, без необхідності постійного опитування (polling) сервера. Використання технології WebSockets дозволяє встановлювати постійне з’єднання між клієнтом і сервером, що дозволяє передавати дані миттєво.
- Обробка подій: Socket.IO дозволяє надсилати події (events) між клієнтом і сервером. Клієнт може відправляти події серверу, а сервер може відправляти події клієнту. Це дозволяє реалізувати взаємодію в реальному часі та спілкування на основі подій між компонентами системи.
- Канали та кімнати: Socket.IO надає можливість створювати канали (channels) та кімнати (rooms), що дозволяє групувати клієнтів і сервери для організації спільних обмінів даними. Клієнти можуть приєднуватись до каналів і кімнат, щоб обмінюватись повідомленнями лише з певною групою клієнтів або серверів.

У більшості вебзастосунків, передбаченні сповіщення користувачів, особливо, якщо вебзастосунок пов’язаний зі сферою надання послуг. Один із способів оповіщення клієнтів, про статус замовлення або іншої інформації, повідомлення на пошту, яку вони вказали при реєстрації, для цього в Node JS існує модуль NodeMailer.

NodeMailer – це модуль для Node.js, який дозволяє відправляти електронні листи (email) з серверної сторони за допомогою протоколу SMTP (Simple Mail Transfer Protocol). Він надає зручний і простий спосіб взаємодії з SMTP-сервером для відправки електронних повідомлень. NodeMailer дозволяє відправляти електронні листи зі своєї Node.js-додатку. Можна встановлювати параметри електронного листа, такі як адреса відправника, адреси отримувачів, тема, текст повідомлення та додатки.

NodeMailer дозволяє додавати вкладення до електронного листа, такі як файли зображень або документи. Ви також можете форматувати текст повідомлення в HTML-вигляді, включаючи стилі, посилання та інші HTML-елементи, саме із-за цієї переваги було вирішено поєднувати NodeMailer разом зі шаблонізатором EJS

EJS (Embedded JavaScript) – це шаблонний двигун (template engine) для Node.js та JavaScript, який дозволяє створювати динамічні HTML-сторінки, використовуючи JavaScript-код для вставки даних та управління розміткою. EJS дозволяє вставляти дані з JavaScript-об’єктів або змінних безпосередньо в HTML-розмітку. Це дозволяє динамічно генерувати контент на стороні сервера і відображати його на стороні клієнта. EJS дозволяє використовувати часткові шаблони (partials), що дозволяє розділяти HTML-розмітку на менші компоненти і використовувати їх у багатьох шаблонах. Це полегшує повторне використання коду та організацію шаблонів.

Тому в парі з nodemailer, який може надсилати html формату повідомлення, можливо створювати повідомлення з даними, якими потрібно та створювати шаблони повідомлень.

Для створення оцінки якості роботи працівників та вебзастосунку, використовуються відгуки, а для створення відгуків чудового підходить Google API, яка надає доступ до Google Form та Google Sheets, з яким можливо витягувати потрібну інформацію – відгуки.

Stripe – це платіжна платформа, яка дозволяє підприємствам та розробникам приймати та обробляти платежі в Інтернеті. Вона забезпечує

інфраструктуру для безпечного збору платежів, переказів коштів та управління платіжними операціями.

Легкість інтеграції: Stripe має документацію та набір інструментів, що дозволяють розробникам швидко і просто інтегрувати платіжну систему в свої вебзастосунки або мобільні додатки.

Безпека: Stripe забезпечує високий рівень безпеки платіжок, включаючи шифрування даних, захист від шахрайства та використання найсучасніших стандартів безпеки.

Управління платіжками: Stripe надає інструменти для керування платіжками, включаючи створення рахунків, відслідковування статусу платіжок, повернення коштів та вирішення спорів.

2.3. Клієнтська частина

Клієнтська частина – це система, яка взаємодіє з користувачем та відображає інформацію на його пристрої. Це та частина програмного забезпечення, яка працює безпосередньо на стороні клієнта. Клієнтська частина вебзастосунку відповідає за створення зручного та привабливого інтерфейсу для користувачів і забезпечує взаємодію з серверною частиною для отримання та обробки даних. Більшу частину успіху клієнтської частини складає користувальницький інтерфейс, а саме на скільки він ефективний та зручний, для користувача.

Інтерфейс вебзастосунку користувача (UI) відіграє найважливішу роль будь-якого вебзастосунку, оскільки він визначає спосіб взаємодії користувача з програмним забезпеченням. Необхідно, щоб користувач зміг швидко і без зусиль вирішити поставлену їм проблему, або ж задовольнити свої потреби, тому створення ефективного вебзастосунку, завжди стоїть у пріоритеті розробника продукту.

Інтерфейс вебзастосунку може мати різні проблеми, які можуть впливати на його зручність використання та ефективність:

Складність та незрозумілість – якщо інтерфейс вебзастосунку має складну структуру або незрозумілі елементи, користувачі можуть мати проблеми

з навігацією та взаємодією. Недостатня відповідність вебзастосунку звичайним вимогам користувачів може стати перешкодою для ефективного використання.

Велика кількість кроків – якщо вебзастосунок вимагає великої кількості кроків для виконання певної дії, це може призвести до втоми та незадоволення користувачів. Довгі форми або складні процеси реєстрації можуть відлякати користувачів і погіршити їх досвід.

Недостатня зрозумілість – якщо інтерфейс не надає достатньої інформації, пояснень або допомоги, користувачі можуть бути збентежені і не знайти потрібну функціональність або не зрозуміти, як користуватися застосунком.

Несприятлива візуальна граматика – недбало підібраний кольоровий спектр, неправильний шрифт, недостатня контрастність або непослідовність у використанні елементів дизайну можуть викликати змішані враження у користувачів і негативно впливати на їх сприйняття та використання застосунку.

Помилки та повідомлення про помилки – якщо інтерфейс не чітко вказує користувачеві про помилки або надає незрозумілі повідомлення про помилки, це може спричинити збентеження та заплутання. Неправильні або загублені дані також можуть впливати на довіру до застосунку та негативно впливати на користувацький досвід.

Відсутність натуральності та потоку – якщо інтерфейс перекладається ускладненими кроками або перериваннями, користувачі можуть втратити потік та зосередження. Недостатня інтуїтивність та неконсистентність у взаємодії можуть спричинити розчарування та зниження задоволення від використання застосунку.

Відсутність персоналізації – якщо інтерфейс не надає можливості персоналізації, користувачі можуть відчувати обмеженість та втрату контролю. Відсутність можливості налаштування інтерфейсу під власні потреби може обмежувати користувачів і знижувати їх задоволення від використання застосунку.

Проблеми такого характеру можуть впливати на загальний досвід користувача та впливати на ефективність та успішність застосунку.

Для досягнення ефективного інтерфейсу для користувача потрібно, варто ретельно аналізувати проблематику вебзастосунків, у сфері для якої вебзастосунок створюється. Також ставити, насамперед не себе, як користувача цим інтерфейсом, а особу, яка повністю не знайома з цим вебзастосунком.

Досягнення ефективності починається з розуміння користувачів, включаючи їх цілі, навички, уподобання та тенденції. Одна з хороших сторін інтерфейсу – це простота. Уникнення непотрібних елементів та чітке формулювання, куди потрібно привернути увагу користувача.

Планування розташування елементів на сторінці з урахуванням їх значення. Ретельне розташування елементів може допомогти привернути увагу до найважливішої інформації, полегшити її пошук та поліпшити читабельність.

Уміле використання кольорів та текстури, може направляти увагу користувача на певні елементи або відводити увагу від інших за допомогою кольору, світла, контрасту та текстур.

Потрібно чітко та ретельно обирати шрифти, використання правильного розміру та стилю тексту допоможе покращити читабельність та зрозумілість.

Забезпечення комунікації, завжди повідомляти користувачів про їх поточне місцезнаходження, виконані дії, зміни стану або помилки. Використання різних елементів інтерфейсу для передачі статусу та, за потреби, наступних кроків може зменшити рівень розчарування для користувача.

Варто підсумувати, що інтерфейс вебзастосунку – одна з ключових переваг, якою може володіти вебзастосунок, тому важливо створити естетично красивий, функціонально ефективний та зручний вебпродукт.

Тому для зручності розробки та створення ефективного і масштабного інтерфейсу користувача, було обрано одну із самих популярних фреймворків, бібліотек – React JS.

React JS – це відкрита JavaScript бібліотека для розробки інтерфейсів користувача. Вона використовується для побудови ефективних та

масштабованих вебдодатків, які забезпечують швидку та динамічну взаємодію з користувачем. Одна з головних переваг ReactJS – віртуальний DOM (Document Object Model). ReactJS створює віртуальне подання сторінки, яке оновлюється ефективно тільки при зміні даних. Це забезпечує високу продуктивність та швидку відповідь на дії користувача.

Замість традиційного імперативного підходу до програмування, де розробник описує кожну деталь взаємодії та зміни стану, ReactJS використовує декларативний підхід. Розробник описує, як повинен виглядати інтерфейс у різних станах, а ReactJS самостійно вирішує, як змінити інтерфейс відповідно до змін стану. ReactJS також пропонує компонентний підхід до побудови інтерфейсів. Вебсторінка розбивається на невеликі та повторно використовувані компоненти, які можна комбінувати разом для створення складних інтерфейсів. Компоненти мають свій власний стан та можуть бути оновлювані залежно від змін вхідних даних або стану додатку.

ReactJS також використовує JSX – розширений синтаксис JavaScript, який дозволяє описувати структуру інтерфейсу в компонентах. JSX поєднує HTML-подібний синтаксис з JavaScript, що робить код більш зрозумілим та підвищує його читабельність.

Axios – це бібліотека JavaScript, яка надає простий та зручний спосіб здійснення HTTP-запитів з веббраузера або з середовища Node.js. Axios дозволяє виконувати різні типи HTTP-запитів, такі як GET, POST, PUT, DELETE, і багато інших. Він надає зручні методи для встановлення заголовків, передачі параметрів запиту, обробки відповідей та перехоплення помилок.

Google API – допомагають імплементувати карти, які надають користувачеві вибір точки А та точки В, більш наглядніше та зручніше.

Lottie – це бібліотека, яка дозволяє відтворювати анімації, створені у програмах для дизайну, таких як Adobe After Effects, на вебсторінках, мобільних додатках та інших платформах. Головною перевагою цієї бібліотеки є те, що вона використовує формат JSON для збереження анімацій та забезпечує плавне

відтворення та інтерактивність анімацій на різних пристроях, з яким дуже легко і зручно працювати в React JS.

Framer Motion – це бібліотека анімації та переходів для React-додатків. Вона надає простий і потужний спосіб створювати різноманітні анімації та ефекти переходів для елементів і компонентів у вашому вебзастосунку. Framer Motion побудований на основі React і добре інтегрується з React-компонентами. Можна легко застосовувати анімації до існуючих компонентів і керувати анімацією з використанням стандартних React-методів життєвого циклу.

РОЗДІЛ 3
ЕТАПИ РОЗРОБКИ

3.1. Розробка серверної частини

Метою є розробка серверної частини вебзастосунку з використанням технології, яка покладена в основу Node JS + Express (для взаємодії з запитами від користувача), а також, забезпечення взаємодії з базою даних MongoDB та серверу, який не тільки працює у режимі реального часу, а й забезпечує роботу в реальному часі та супроводжується відгуками клієнтів, які зберігаються у базі даних.

Далі підключення бази даних. Для роботи з базою MongoDB застосовується бібліотека mongoose, яка взаємодіє з базою через технологію mongooseSchema – опис полів та їх значень об’єкта для імплементації його в базу, метод – модель об’єкта, схема об’єкта. Усі моделі зберігаються в окремій папці models.

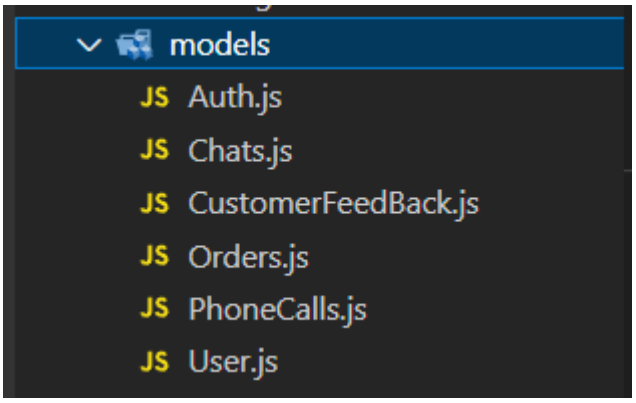

Рис. 3.1. Моделі для об’єктів, які будуть додаватися в базу даних

Приклад опису моделі:

```
const mongoose = require('mongoose')
const Schema = mongoose.Schema

const authusersSchema = new Schema({
  email:{
    type: String,
    required: true
  },
  password:{
    type: String,
```

```
        required: true
    },
    info: {
        name: {
            type: String
        },
        phone : {
            type: String
        },
        address: {
            type: String
        }
    },
    discount: {
        type: Number
    },
    role: {
        type: String,
        required: true
    }
}, {timestamps: true})

const Authusers = mongoose.model('Authusers', authusersSchema)

module.exports = Authusers
```

Після підключення до бази, створення слухача за змінами у колекції, для визначення змін, які відбуваються в реальному часі

```
const userColection = client.db().collection('users')
let changesStreamUsers = userColection.watch()
const conector = mongoose.connection
```

Приклад слухача на зміни в колекції

```
    changeStream.on("change", async change => {
        if (change.operationType == 'insert') {
            const newOrder = {... changes.fullDocument}
            const result = newOrder
            console.log('next', result);
            check(result, 'changeStatus')
        } else if (change.operationType == 'update') {
            const result = await ordersColection.findOne({ _id: change.documentKey._id })
            if (change.updateDescription.updatedFields.status == 'done') {
```

```
                doneOrder(result._id, result.owner)
            } else if (change.updateDescription.updatedFields.status == 'active') {
                activeMessage(result)
            }
            console.log(change)
            check(result, 'updatedOrder')
        }
    });
```

Також застосовані методи відправки повідомлень, за допомогою NodeMailer doneOrder(), якщо замовлення виконано та activeMessage(), якщо замовлення в стані активності.

Для реалізації відповідей на запити клієнтів використовується Socket.IO за допомогою нього функція check() перевіряє чи адміністратор онлайн, якщо так – надсилає об'єкт, який було змінено.

```
    async function check(changes, method) {
    const first = await authColection.findOne({ email: changes.owner })
    first == null ? '' : socketIO.to(first.socketId).emit(`${method}`, changes)
    const adminCheck = await authColection.findOne({ email: 'Admin' })
    adminCheck == null ? '' : socketIO.to(adminCheck.socketId).emit(`${method}`,
changes)
}
```

Для взаємодії з відгуками клієнтів, використовувалися Google API's, а саме бібліотеки, за допомого яких можна отримати доступ до Google Sheets. Як імплеменувати API's у серверну частину можна дізнатися через документації по джерелу.

За допомогою loadInfo() загрузити документ, та обрати потрібні дані

```
    const sheet = doc.sheetsByIndex[0]
        const rows = await sheet.getRows()
        for (let i = 0; i < rows.length; i++) {
            let review = {
                id: rows[i].ID,
                owner: rows[i].Name,
                rate: rows[i].Rate,
                review: rows[i].Review,
                time: rows[i].TimeStamp
            }

            console.log(review)
```

Відгуки, які не мають id повинні додаватися до бази даних, як нові.

```
            if (review.id == undefined) {
                const result = new CustomerFeedBack({ owner: review.owner, rate: review.rate, review: review.review, time: review.time })
                result.save()
                rows[i].ID = 'added'
                await rows[i].save()
            }
        }
    }
```

Також створений таймер для подальшого виклику функції через деякий час, якщо з'явилися нові відгуки.

```
    setTimeout(authSheets, 36000000)
```

При роботі зі Stripe потрібно не забувати головне, що ціна вимірюється в центах тому, якщо використовувати іншу валюту, потребується додаткова конвертація.

Для того, щоб надати клієнту можливість оплатити продукт у Stripe вмонтована функція створення платежу – paymentIntents.create(), вона приймає достатньо велику кількість даних, з якими можна ознайомитись в документації, в цьому випадку використовуються поля:

- Description – для передачі Id
- Amount – кількість товару
- Currency – валюта
- Payment_method_types – методи оплати

```
        async function createPayment(order) {
            let priceInCent = await Math.round((order?.price / 36.7) * 100)
            if(priceInCent == NaN) {
                priceInCent = await Math.round((order?.price / 36.7) * 100)
            }
            console.log('priceInCent -> ', priceInCent)
            const paymentIntent = await stripe.paymentIntents.create({
                description: _id,
                amount: priceInCent,
                currency: "usd",
                payment_method_types: ['card'],
            });
```

```
    res.send({
      paymentIntent
    });
  }
```

3.2. Розробка клієнтської частини

Метою є розробка клієнтської частини вебзастосунку. Для цього було обрано за основу React JS – система, бібліотека або ж фреймворк, який в рази полегшує розробку та надає за допомогою JSX-файлів умови, при яких можливо зручно і коротко користуватися HTML і JavaScript одночасно.

Потрібно згадати, особливості React JS:

- Підтримка JSX-файлів – одночасно може виконувати HTML і JavaScript код (код JS вбудовується через синтаксис фігурних дужок - {})
- React компонентонаправлений, тобто усі сторінки представляються у вигляді компонентів, які можна імпортувати. Компонент може мати в собі безліч компонентів. Компонент – це функція, яка експортується export default function <Назва функції>()
- Основа частина коду починається з export default function <Назва функції>(){
  return(
)
  }

3.2.1. Роутинг вебзастосунку

React JS популярний не тільки тим, що він надає, дуже зручні інструменти для розробки вебзастосунків, але й тим, що застосунки на його основі працюють в режим Single Page APP – зміни на сайті відбуваються без перезавантаження. Тому для роутингу на вебзастосунку використовується інструмент useNavigate(), в ньому вказується url-адресс, куди потрібно перейти, а він в свою чергу дає сигнал React, який компонент завантажити. Для роутингу також потрібні такі інструменти, як – react router dom, який містить в собі компоненти BrowserRouter, Routes, Route для роутингу на вебзастосунку

Файл роутингу

```
export default function Router({socket, isLoaded}) {
  return(
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<HomePage socket={socket} isLoaded={isLoaded}/>}>
          <Route path="home" element={<WelcomePage/>} />
          <Route path="aboutUs" element={<AboutUs />} />
          <Route path="advantages" element={<Advantages />} />
          <Route path="offers" element={<Offers />} />
          <Route path="trucks" element={<Trucks />} />
          <Route path="partnes" element={<Partners />} />
          <Route path="how" element={<How />} />
          <Route path="form" element={<Form />} />
        </Route>
        <Route path="/userPage" element={<UserPage isLoaded={isLoaded} socket={socket}/>}/>
                <Route path="/userPage/User/order/:id" element={<UserOrder socket={socket}
isLoaded={isLoaded}/>}/>
      </Routes>
    </BrowserRouter>
  )
}
```

3.2.2. Глобальні змінні

Перед реалізацією головної частини, потрібно ініціалізувати декілька глобальних змін, які будуть використовуватися у компонентах застосунку:

Current – описує користувача

Глобальні змінні для того, щоб при потребі виконувати маніпуляції з користувачем, не перенаправляти змінну по всім компонентам через props

```
export const CurrentContext = createContext()

const clearData = {
  id: ‘‘,
  email: ‘‘,
  password: ‘‘,
  info: {
    name: ‘Guest’,
    phone: ‘‘,
    address: ‘‘
```

```
    },
    discount: ‘‘,
    role: ‘‘
}

const CurrentProvider = ({ children }) => {
    const [current, setCurrent] = useState(clearData)

    return (
        <CurrentContext.Provider value={{ current, setCurrent }}>
            {children}
        </CurrentContext.Provider>
    )
}

export default CurrentProvider
```

3.2.3. Кабінет користувача та адміністратора

Кабінет користувача повинен бути доступний відразу після реєстрації або входу в обліковий запис. Сторінка користувача знаходиться по новому посиланню тому, для початку потрібно перенаправити його за допомогою інструменту useNavigate().

```
const navigate = useNavigate();
        click: function () {
            navigate(`/userPage`)
        }
```

Переадресація відбувається на посилання – /userPage, за який відповідає компонент UserPage – головний компонент, файл в якому буде вирішуватися, чи авторизований користувач: адміністратор, працівник чи клієнт.

```
current.role !== ‘Admin’ ? <UserMain isLoaded={isLoaded} socket={socket}/> : <Admin isLoaded={isLoaded} socket={socket}/>
```

Компонент UserMain відповідає за кабінет користувача, компонент Admin відповідає за кабінет адміністратора, працівника. Компонент UserMain повинен містити в собі усі компоненти, які відображають інтерфейс кабінету.

3.2.3.1. Історія замовлень

Містить в собі усі замовлення користувача. Для початку створюється за допомогою бібліотеки AXIOS, Post – запит, який приймає в себе об’єкт, який

містить в собі дані користувача. Після успішного отримання даних від серверу записує отриману інформацію у зміну, стан History.

```
const [history, setHistory] = useState([])
useEffect(() => {

    axios.post('/userPage/orders', { ...current }).then(res => setHistory([...res.data]))
}, [])
```

Далі за допомогою методу map, проходить через усі елементи history та відображає їх на сторінці.

Усі елементи представляються у вигляді кнопок, щоб клієнт (користувач) міг натиснути на елемент, та йому відображалась інформація про замовлення, яке його цікавить. Для цього створено стан, зміну show, яка буде змінюватися при натисненні на кнопку і буде відображати потрібне замовлення замість списку.

```
const [show, setShow] = useState(false)
    <div className={styles.items}>
      <h2>Історія замовлень</h2>

      <div className={styles.item}>
        {
          history.map(item => (
            show ?
```

При значенні false відображається історія замовлень, при true з історією зникає за допомогою css display

```
style={{ display: show ? 'none' : 'block' }}
```

та відображається замовлення, яке потрібно.

При натисненні створюється post – запит на сервер для отримання замовлення, яке потрібне та записується у змінну, стан order

```
const [order, setOrder] = useState()
  function getId(e) {
    e.preventDefault()
    setGetId(e.currentTarget.id)
    axios.post('/userPage/getOrderByID', { id: e.currentTarget.id }).then(res => setOrder({ ...res.data }))
    setShow(true)
  }
```

Після чого замовлення відображається на панелі користувача. Якщо замовлення не перебуває в стані – active, то в процесі відображається інтерфейс, який надає змогу користувачу внести зміни в замовлення.

```
<div className={styles.item}>
  {
      history.map(item => (
    show ?
      <div style={{ display: show ? 'block' : 'none' }}>
        {
          item._id == getid ? <div>
            {
              item.status == 'active' || item.status == 'done' ?
```

Якщо замовлення має статус – active, то в процесі користувачу надається інтерфейс, який надає можливість, тільки переглядати замовлення.

3.2.3.2. Створення замовлень

Створення замовлення в кабінеті клієнта відбувається в компоненті CreateOrder.

При створенні форми для замовлення використовуються Google Api's, ComboBox та створення запиту на оплату в системі Stripe.

Для зручного вибору точки А, точки відправлення, та точки В, точки призначення, використовуються Google API's, а саме Goolge Maps. Все імпортується з бібліотеки react google maps/api.

```
import { DirectionsRenderer, GoogleMap, Marker, Polyline, useLoadScript } from "@react-google-maps/api";
```

Завантаження карти відбувається в головному файлі, компоненті App та передається через props в цей – CreateOrder компонент. Карта представляється компонентом, який містить в собі декілька параметрів для його відображення та взаємодії, а саме:

- mapContainerClassName – клас, який відповідає за розміри. Якщо не задавати розмір, то карта не буде відображатися.
- center – куди буде вказувати карта за замовчуванням.
- zoom – масштаб зуму.

```
<div style={{
    display: 'flex', alignItems: 'center'
}}>
    {!isLoaded ? (
        <h1>Loading...</h1>
    ) : (
        <GoogleMap
            mapContainerClassName={styles.mapContainer}
            center={firstMarker.lat !== '' ? firstMarker : center}
            zoom={12}
        >
```

Для створення маркерів на карті, які будуть вказувати на потрібні точки в бібліотеці, використовується компонент Marker, а маркери будуть зберігатися у змінній – markers.

```
const [markers, setMarkers] = useState([])
```

```
{
    markers.map(item => (
        <Marker position={item} />
    ))
}
```

А для створення маркерів використовується додаткова бібліотека use places aoutocomplete, для отримання місця та його географічний параметрів

```
import usePlacesAutocomplete, { getGeocode, getLatLng } from 'use-places-autocomplete'
```

Функція-компонент MarkerInput буде відповідати за встановлення користувацького маркеру. В ньому використовується Combobox бібліотека, яка дозволить отриману інформацію від use places aoutocomplete висвітлювати як випливаюче вікно.

Також надається можливість використати адресу по замовчуванню, якщо потрібно. Після вибору місця інформація, яка зберігається в data переходить в функцію утримувач та конвертується в географічні координати.

```
const handleSelect = async (address) => {
  setValue(address, false)
  clearSuggestions()

  const results = await getGeocode({ address })
  const { lat, lng } = await getLatLng(results[0])
  console.log(results)
```

```
        setDeleteCordinats({ lat, lng })
        setMarker({ lat, lng })
        first ? order.adress.from = address : order.adress.to = address
    }
```

Географічні координати передаються у змінні firstMarker та secondMarker через props.

Для підрахування дистанції між точками використовується GoogleMaps DirectionsService(), він створює запит на сервер і повертає дистанцію. Функція спрацьовує після заповнення змінної secondMarker. В роут, який створює DirectionsService, потрібно передати 3 параметри:

- origin – точка A
- destination – точка B
- travelMode – спосіб транспортування

```
        const directionsService = new google.maps.DirectionsService()
        const results = await directionsService.route({
          origin: firstMarker,
          destination: secondMarker,
          travelMode: google.maps.TravelMode.DRIVING
        })
        console.log(results)
        setDirectionResponse(results)
        setDistanceF(results.routes[0].legs[0].distance.value)
        setDuration(results.routes[0].legs[0].duration.text)
```

Після чого дистанція, яка буде відображатися на карті записується у змінну – directionResponse, а числове представлення у змінну distanceF. Далі відображається дистанція між точками на карті.

Далі, після заповнення поля Вага відбувається розрахунок ціни перевезення, з урахування знижки чи без.

```
    useEffect(() => {
     +order.weight
          order.weight == 0 || order.weight == '' ? setPrice(0)
      : order.weight <= 1 ? setPrice((distanceF / 1000) * 3)
      : order.weight >= 1 && order.weight < 3 ? setPrice((distanceF / 1000) * 5)
      : order.weight >= 3 && order.weight < 5 ? setPrice((distanceF / 1000) * 7)
      : order.weight >= 5 && order.weight < 7 ? setPrice((distanceF / 1000) * 12)
    : order.weight >= 7 && order.weight <= 10 ? setPrice((distanceF / 1000) * 15) : ''
```

```
        setDiscount({ discount: current?.discount })
        if (discount.discount > 0) {
            +price
            setPriceWithDiscount(price - (price * (discount.discount * 100) / 100))
        }
    }, [price, discount.discount, order.weight, discount.discount])
```

Якщо користувач погоджується з ціною і натиска кнопку: „Створити замовлення” форма з створення замовлення ховається і відображається вибір:

- Перейти на замовлення
- Оплатити

Якщо користувач натиснув кнопку Перейти на замовлення, відбувається перенаправлення на сторінку зі створеним замовленням.

Якщо натиснув оплатити, то ініціалізується функція payment, в якій створюється запит на сервер для створення платежу в системі Stripe.

```
    useEffect(() => {
        payment ? axios.post('/payment', { orderId: paymentId, _id: current._id, })
        .then(res => (console.log(res), setClientSecret(res.data.paymentIntent.client_secret),
        setHandleId(res.data.paymentIntent.description), setShowForm(true))) : ''
    }, [payment])
```

Після отримання відповіді від сервера ClientSecret – змінна, для синхронізації форми для оплати на клієнтський частині записується у змінну clientSecret, id замовлення записується у змінну handleId та показується форма оплати.

Для показу форми оплати Stripe використовується компонент CheckOutForm, який імпортується з бібліотеки Stripe.

Після оплати йде переадресація на сторінку з замовленням.

3.2.3.3. Редагування замовлення

Обирається поля, які потрібно змінити та заповнюються. Якщо поле змінювати не потрібно, то функція заповнює пробіли інформацією замовлення, яка вже є в базі даних.

```
    function orderChanges(e) {
        e.preventDefault()

        changes.status = status
```

```
        priceWithDiscount > 0 || testprice ? changes.price = priceWithDiscount
          : price > 0 ? changes.price = price.price : ‘‘
        testprice > 0 ? changes.price = testprice : ‘‘

        for (let element in changes) {
          if (changes[element] == ‘‘) {
            if (element == ‘from’ || element == ‘to’) {
              for (let another in order.adress) {
                console.log(‘wow’)
                console.log(order.adress)
                if(element == another) {
                  changes[element] = order.adress[another]
                }
              }
            } else {
              changes[element] = order[element]
            }
          } else {
            changes[element]
          }
        }

        console.log(‘result ->‘, { ...changes })
        axios.put(‘/userPage/changeOrder’, { ...changes, adress: {from: changes.from, to: changes.to}})
        .then(res => (setOrder({ ...res.data }, setChanges(clearData),
        setPriceWithDiscount(‘‘), setDiscount(‘‘))))
        setChanges(clearData)
      }
```

3.2.3.4. Чат

Повідомлення приходять від серверу у режимі реального часу за допомогою socket події newMessage.

```
      useEffect(() => {

        socket.on(‘newMessage’, data => {
          let newMessage = {}

          for (let element in data) {
            newMessage = data[element]
          }
```

```
            setChatHistory(prev => [...prev, { ...newMessage }])
        })

        return () => {
            socket.off('newMessage')
        }
    }, [socket])
```

На сервері встановлено слухач змін бази даних. Тому, якщо відбувається зміна в полі чату, то відбувається імітування події newMessage

```
            console.log('changes\n', change.updateDescription)
            const result = await chatsColection.findOne({ _id: change.documentKey._id })
            if (change.updateDescription.updatedFields.Admin) { return } else {
socketIO.to(result.id).to(result.Admin).emit('newMessage', change.updateDescription.updatedFields)
```

Для надсилання повідомлення використано функцію postMessage, вона імітує подію створення нового повідомлення

```
    function postMessage(e) {
        e.preventDefault()

        socket.emit('postMessage', { ...message, chatId: getid, owner: 'Admin', sokcetId: socket.id })
        setMessage(clearData)
    }
```

На серверній частині

```
        socket.on('postMessage', data => {
            console.log('post data ->', data)
            const result = chatsColection.findOneAndUpdate({ _id: new BSON.ObjectId(data.chatId) }, {
                $push: {
                    messages: { message: data.message, id: data.id, owner: data.owner }
                }
            }, { new: true })
        })
```

РОЗДІЛ 4
ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1. Вступ

Мета полоягає в онайомленні з головним функціоналом користувача на вебзастосунку, взаємодію його з інтерфейстом користувача та сервером.

4.2. Головна сторінка

При переході на вебзастосунко користувача зустрічає головна сторінка та панель навігації, за допомогою яких можна здійснювати переходи між ключовими пунктами інтерфейсу та користувача, також при навігації присутні анімації появи, для більш естетичної навігації. (рис. 4.1 – 4.3)

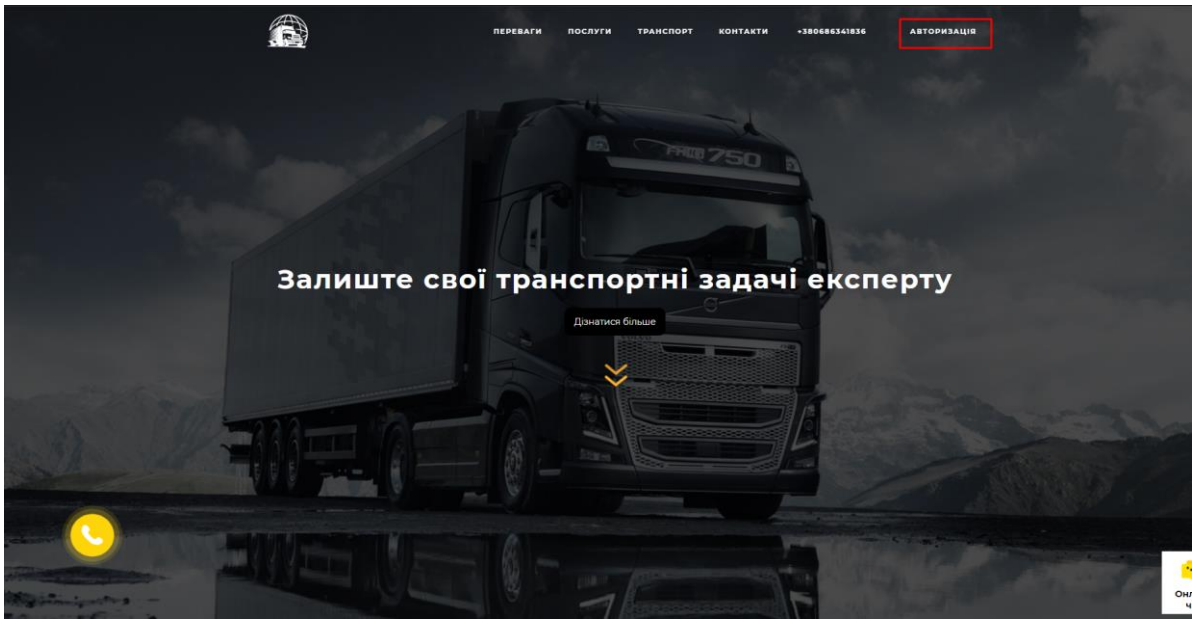

Рис. 4.1. Головна сторінка

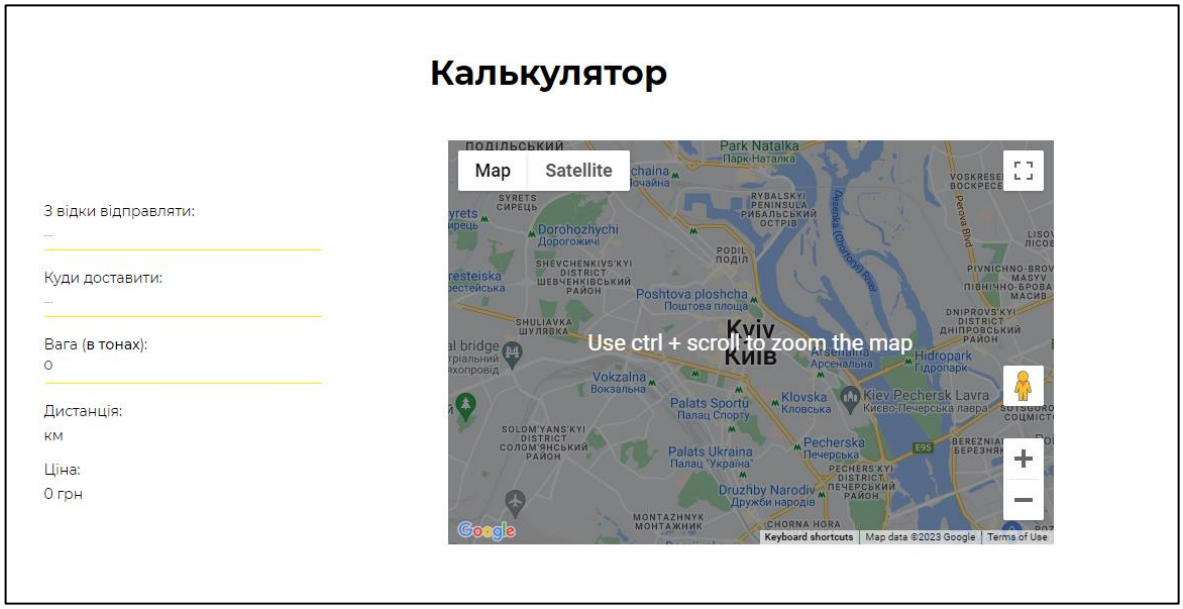

Рис. 4.2. Калькулятор

Рис. 4.3. Переваги

4.2. Авторизація

Доступ до авторизації відбувається при натисненні на кнопку: «Авторизація», яка знаходиться на Header-і вебзастосунку. Після чого з’являється меню вибору авторизації та заповнюються усі поля. (рис. 4.4 – 4.5)

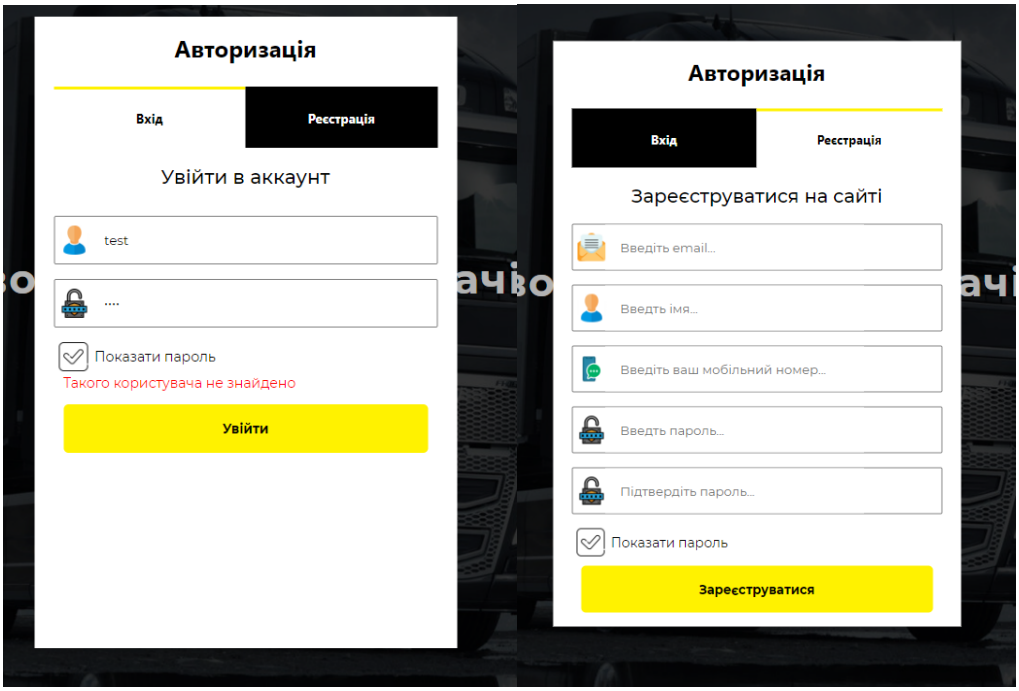

Рис. 4.4. – 4.5. Авторизація та Реєстрація

При успішній реєстрації відбувається перенаправлення користувача на головну сторінку і зверху на header-і сторінки буде відображатися його ім’я замість кнопки Авторизація (рис. 4.6)

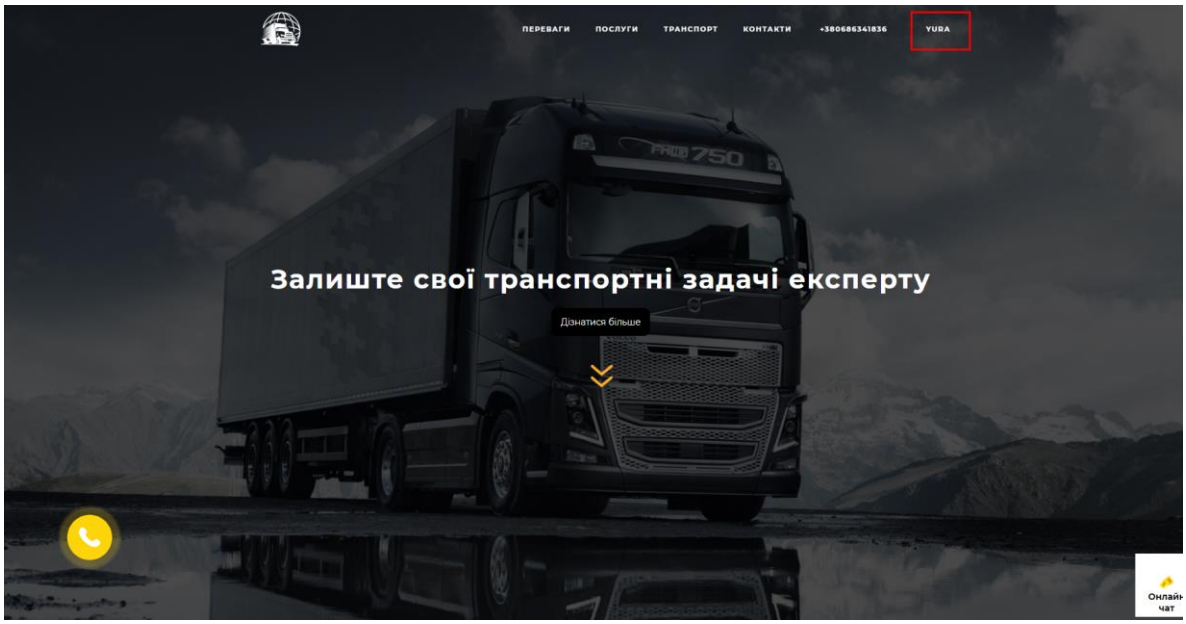

Рис. 4.6. Успішна авторизація

Та на вказану електронну пошту прийде повідомлення про успішну реєстрацію. (рис. 4.7)

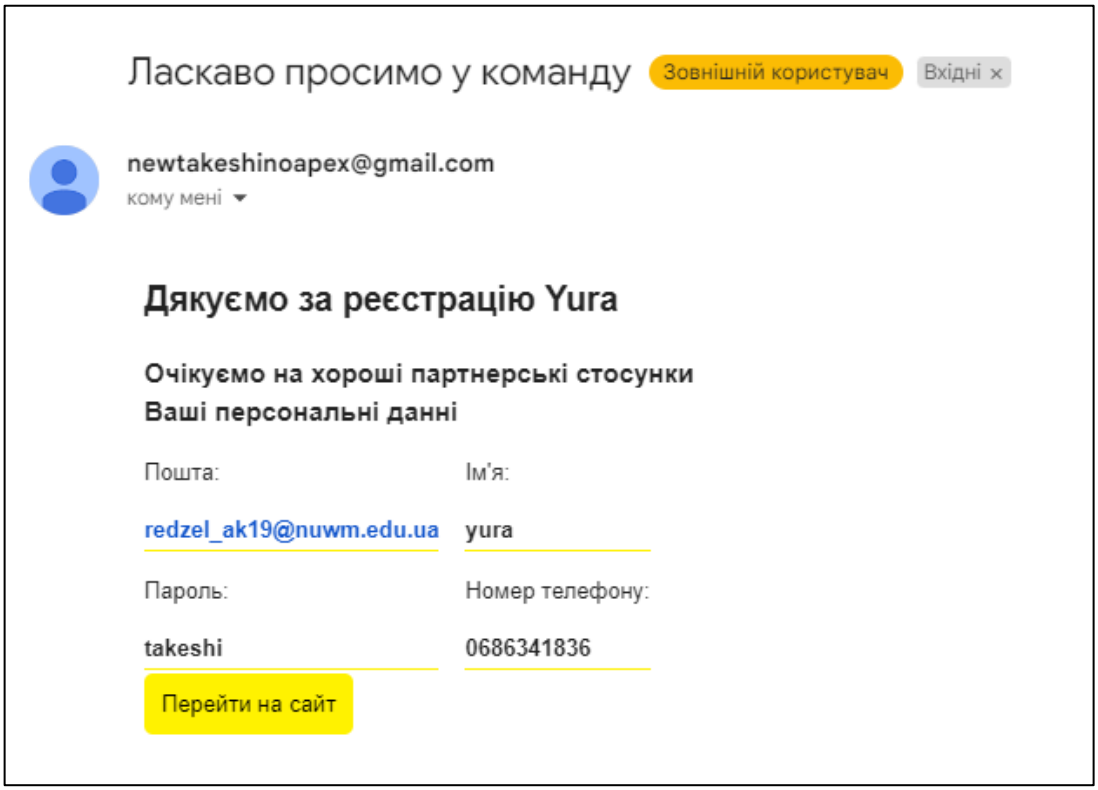

Рис. 4.7. Повідомлення на електронній пошті про реєстрацію

4.3. Створення замовлення

Створення замовлення відбувається у кабінеті користувача. Для переходу в нього потрібно авторизуватися та після успішної авторизації з’явиться кнопка з іменем користувача замість кнопки «Авторизація». Потрібно натиснути на кнопку з іменем користувача та в меню, яке з’явиться, обрати пункт: «Ваш аккаунт» (рис. 4.8)

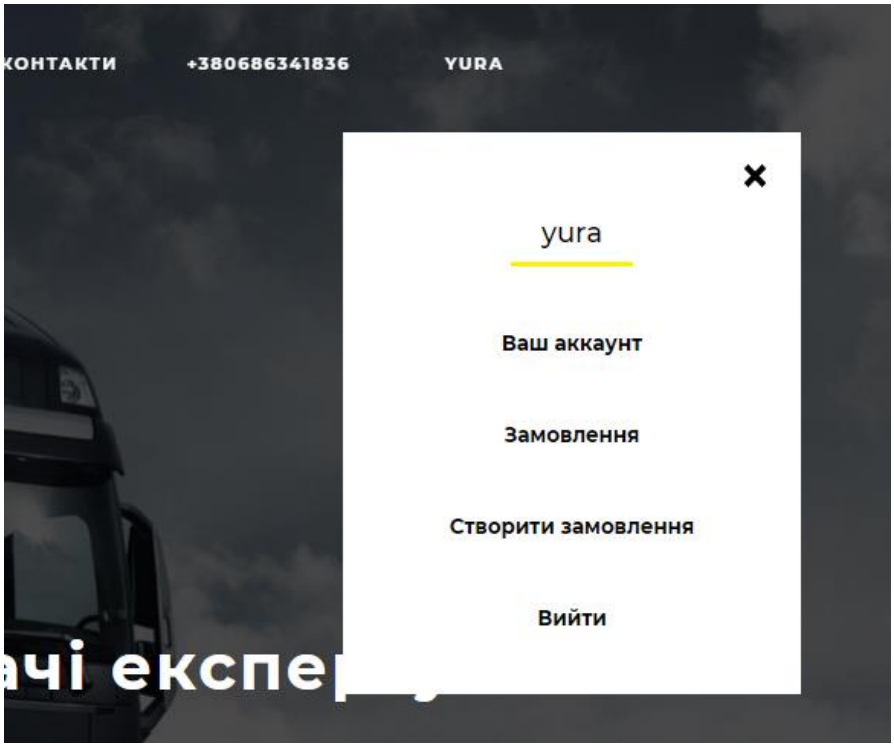

Рис. 4.8. Навігація користувача на головній сторінці

Після чого користувач направляється в свій особистий кабінет. (рис. 4.9)

Рис. 4.9. Кабінет користувача

Для створення замовлення потрібно перейти у відповідне меню натиснувши кнопку: «створити замовлення» (рис. 4.10 – 4.11)

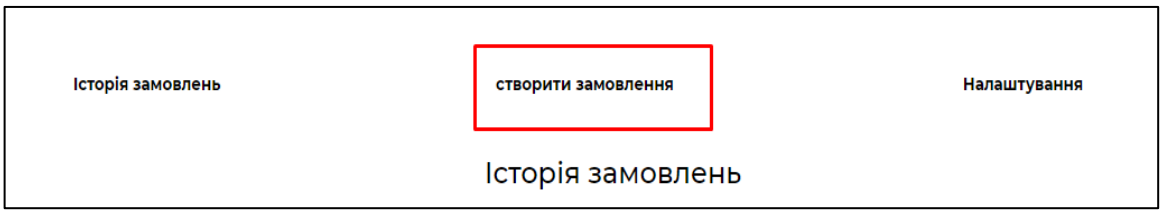

Рис. 4.10. Кнопки навігації

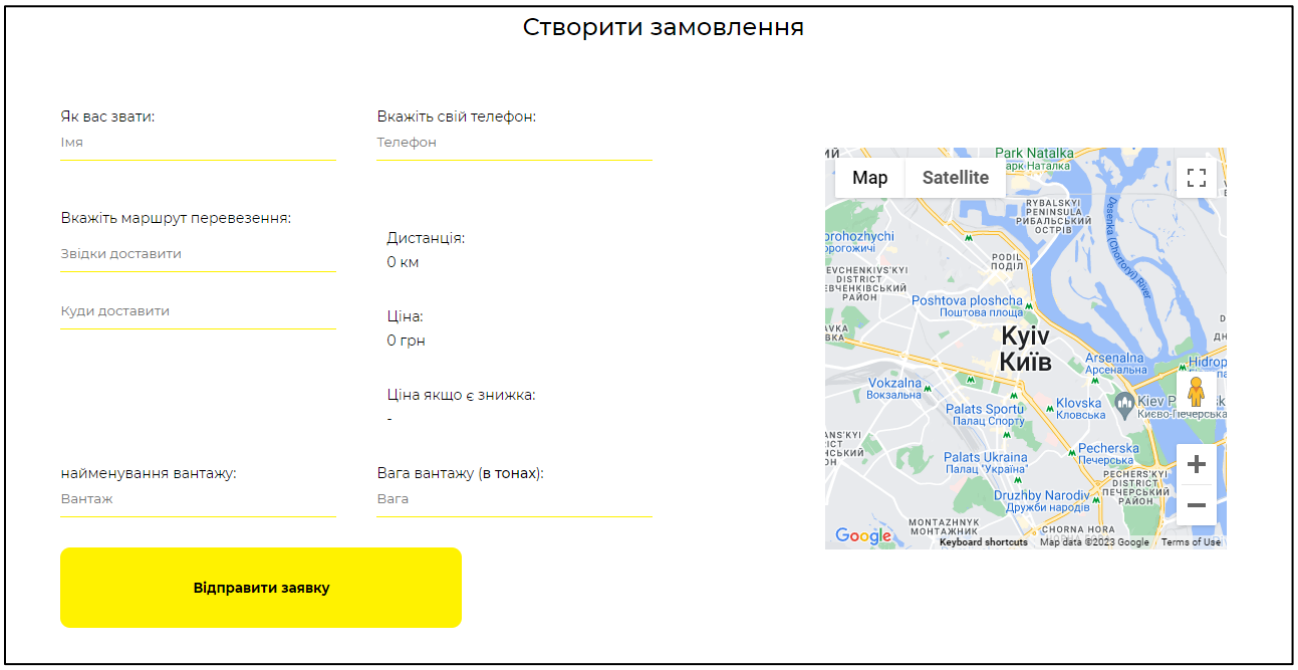

Рис. 4.11. Меню створення замовлення

Після чого заповнити усі поля, які потрібні для створення замовлення. У випадку, якщо користувач не заповнив усі необхідні поля, виникне повідомлення-попередження. (рис. 4.12)

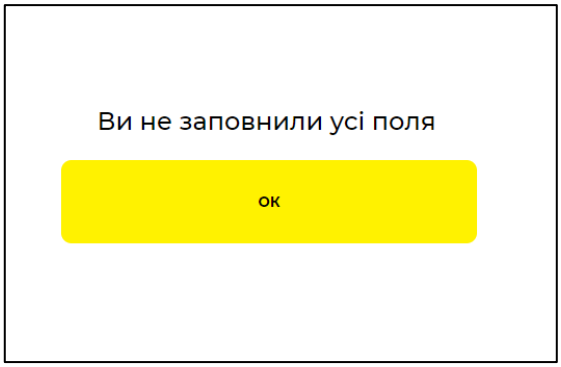

Рис. 4.12. Повідомлення про те, що користувач не заповнив усі поля

При заповненні усіх необхідних полів, якщо у користувача не було до цього адреси за замовчуванням, буде запропоновано додати її як основну.

Після чого відбувається перенаправлення на вибір оплати, чи оплачувати замовлення, чи ні. Якщо користувач відмовиться оплачувати, то його буде перенаправлено на сторінку зі своїм замовлення, у протилежному випадку з’явиться форма для оплати і при успішній оплаті, його замовлення перейде в активний стан та перенаправить на сторінку з його замовленням. (рис. 4.13 - 4.15)

Рис. 4.13. Меню вибору дії, після заповнення полів

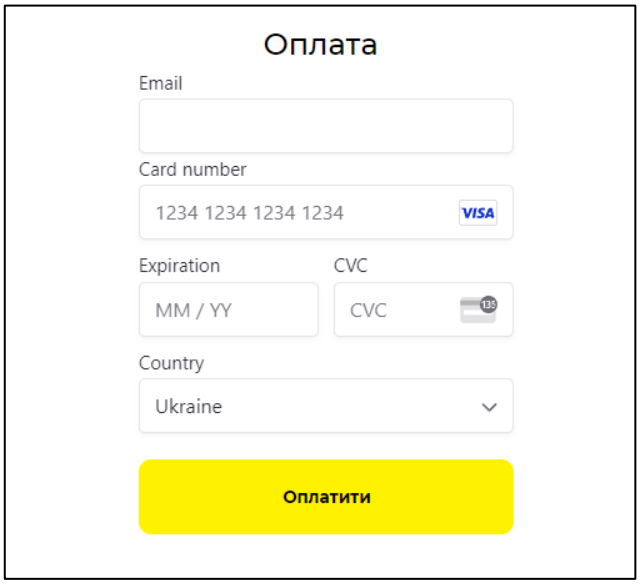

Рис. 4.14. Оплата

Також буде відправлено повідомлення на електронну пошту про успішне створення замовлення. (рис. 4.15)

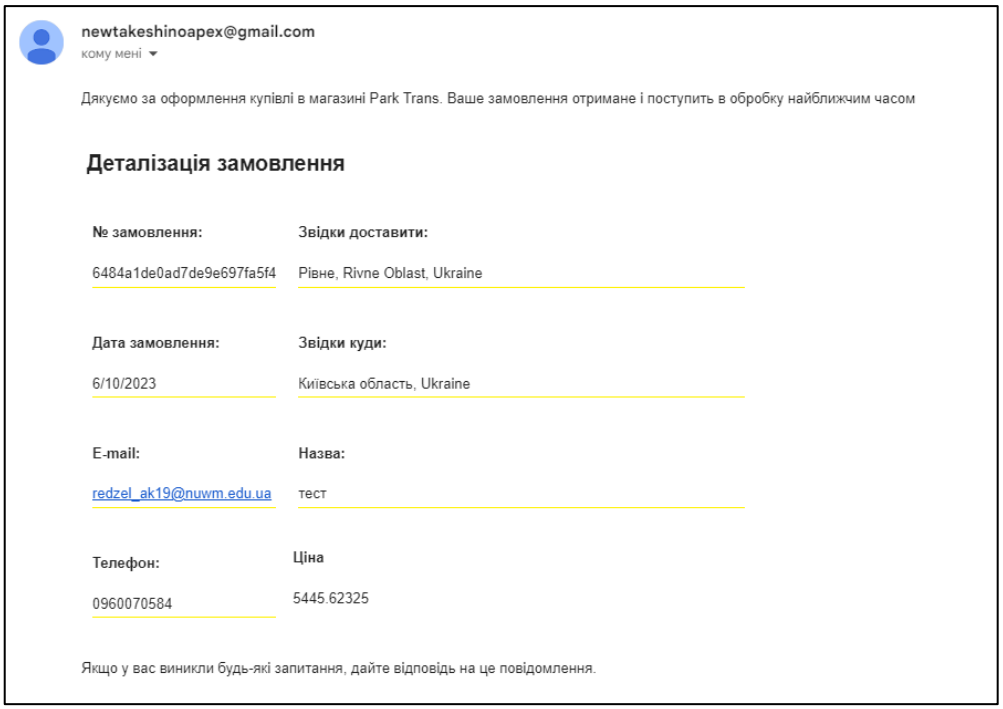

Рис. 4.15. Повідомлення, про створення замовлення на пошті

4.4. Редагування замовлення

Редагування замовлення відбувається в пункті історія замовлень. Якщо замовлення не в стані активності, то є можливість змінити його (рис. 4.16)

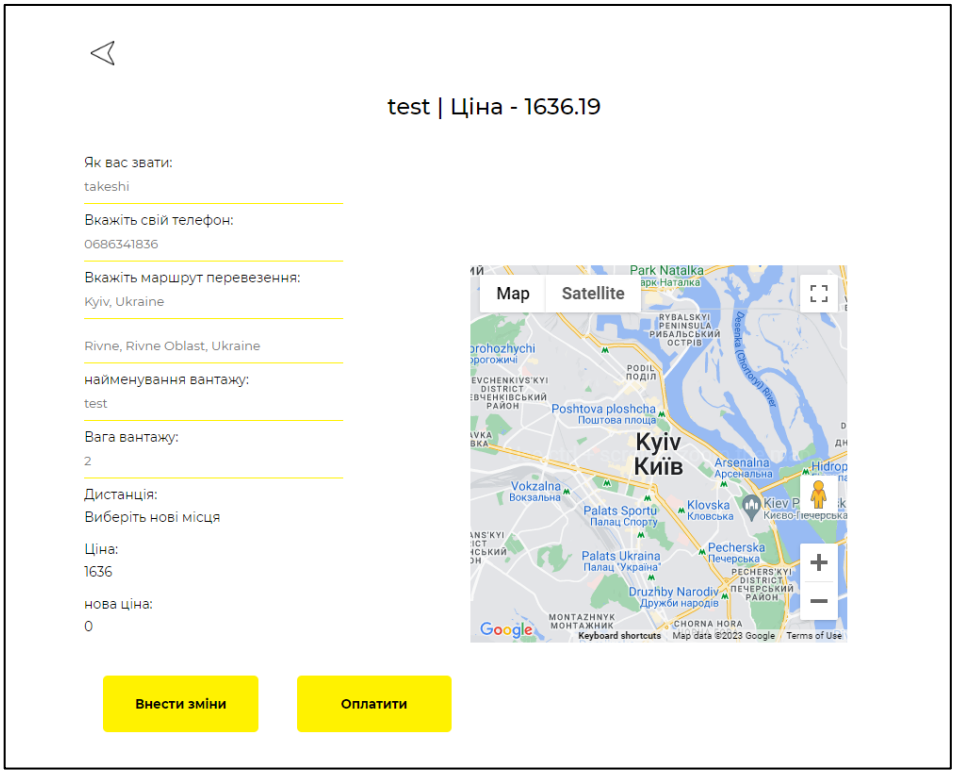

Рис. 4.16. Повідомлення, про створення замовлення на пошті

ВИСНОВОК

У результаті виконання кваліфікаційної роботи було розроблено вебзастосунок для транспортних перевезень, який передбачає функціонально-ефективний та естетично привабливий інтерфейс користувачу та забезпечує комунікацію між працівником та клієнтом в режимі реального часу.

Протягом виконання поставленого завдання – створення вебзастосунку, було поставлено відповідну мету, яка передбачала відповідні завдання для реалізації вебзастосунку, які включали створення серверної частини та клієнтської частини. Провівши аналіз, за основу серверної частини було взято Node JS та бібліотеки, фреймворки, які надають зручні елементи для реалізації серверної частини, а також налагодження легкої та швидкої взаємодії з базою даних. За основу взято React JS – інструмент для зручного і менш об’ємного використання HTML та Java Script коду, що полегшує використання технологій, які надають можливості створювати унікальні, динамічні та естетично красиві вебінтерфейси з урахуванням швидкого і зручного створення запитів на сервер та комунікації клієнтської частини з серверною.

У процесі реалізації та тестування було повністю втілено всі функціональні та естетичні завдання відповідно до вимог, які були покладені на створення вебзастосунку. Шляхом постійного тестування були виправлені усі функціональні та візуальні дефекти.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Node.js MongoDB [електронний ресурс] - https://www.w3schools.com/nodejs/nodejs_mongodb.asp
2. Mongoose v7.3.0: Schemas [електронний ресурс] - https://mongoosejs.com/docs/guide.html
3. Getting Started with Google Sheets API in Node.js [електронний ресурс] - https://www.section.io/engineering-education/google-sheets-api-in-nodejs/
4. Accept a payment | Stripe Documentation [електронний ресурс] - https://stripe.com/docs/payments/accept-a-payment
5. How To Accept Payments With Stripe [електронний ресурс] - https://www.youtube.com/watch?v=1r-F3FIONl8&t=545s&ab_channel=WebDevSimplified
6. Google-spreadsheet - npm [електронний ресурс] - https://www.npmjs.com/package/google-spreadsheet
7. Control Google Sheets with Node.js / JavaScript (2021) [електронний ресурс] - https://www.youtube.com/watch?v=PFJNJQCU_lo&t=951s&ab_channel=JamesGrimshaw
8. NODEMAILER [електронний ресурс] - https://nodemailer.com/about/
9. Introduction | Socket.IO [електронний ресурс] - https://socket.io/docs/v4/
10. Building a chat app with Socket.io and React [електронний ресурс] - https://dev.to/novu/building-a-chat-app-with-socketio-and-react-2edj
11. React Router [електронний ресурс] - https://www.w3schools.com/react/react_router.asp
12. Integration of Google Maps with React [електронний ресурс] - https://medium.com/scalereal/integration-of-google-maps-with-react-part-1-86c075ab452a
13. How to load Maps JavaScript API in React [електронний ресурс] - https://www.youtube.com/watch?v=9e-5QHpadi0&ab_channel=GoogleMapsPlatform

14. How to add Place Autocomplete input to a map in React [електронний ресурс] - https://www.youtube.com/watch?v=BL2XVTqz9Ek&ab_channel=GoogleMapsPlatform
15. React Stripe.js reference [електронний ресурс] - https://stripe.com/docs/stripe-js/react
16. How To Use Lottie Animations in a React App – LottieFiles [електронний ресурс] - https://lottiefiles.com/blog/working-with-lottie/how-to-use-lottie-in-react-app
17. Framer-motion [електронний ресурс] - https://www.npmjs.com/package/framer-motion

ДОДАТКИ

Додаток А. Код серверу

```
const express = require('express')
const mongoose = require('mongoose')
const { MongoClient } = require('mongodb')
const methodOverride = require('method-override')
const stripe = require('stripe')
const app = express()
const http = require('http');
const server = http.createServer(app);
const { Server } = require("socket.io");
const io = new Server(server);
const PORT = 5000
const cors = require('cors')
const socketIO = require('socket.io')(server, {
    cors: {
        origin: "http://localhost:5173"
    }
});
app.use(cors())
app.use(express.json())
app.use(express.urlencoded({ extended: false }))
app.use(methodOverride('_method'))
const client = new MongoClient()
mongoose.connect(database, { useNewUrlParser: true, useUnifiedTopology: true }).then((res) =>
console.log('Connected to DB')).catch((error) => console.log(error))
const userColection = client.db().collection('users')
const ordersColection = client.db().collection('orders')
const onlineColection = client.db().collection('onlineUsers')
const authColection = client.db().collection('authusers')
const phoneColection = client.db().collection('phonecalls')
const chatsColection = client.db().collection('chats')
const feedBackColection = client.db().collection('customerfeedbacks')
let changeStream = ordersColection.watch()
let changesStreamUsers = userColection.watch()
let changesStreamOnlineUsers = onlineColection.watch()
let changesStreamAuthUsers = authColection.watch()
let changesStreamPhoneCalls = phoneColection.watch()
```

```
let changesStreamChats = chatsColection.watch()
const conector = mongoose.connection
module.exports = {
    userColection, ordersColection,
    onlineColection, authColection, phoneColection,
    changeStream, changesStreamUsers, changesStreamOnlineUsers,
    changesStreamAuthUsers, changesStreamPhoneCalls, changesStreamChats,
    chatsColection, stripe, feedBackColection,
    conector, socketIO, client
}
server.listen(PORT, () => {
    console.log('listening on *:5000');
});
const authorizationRoutes = require('./routes/authorization-routes')
const callRoutes = require('./routes/call-routes')
const chatRoutes = require('./routes/chat-routes')
const feedBackRoutes = require('./routes/feedBack-routes')
const ordersRoutes = require('./routes/orders-routes')
const usersRoutes = require('./routes/users-routes')
const authSheets = require('./feedBack/feedBack')
const {socketIO_logic, streams_logic} = require('./socket.io/socket-logic')
app.use(authorizationRoutes)
app.use(callRoutes)
app.use(chatRoutes)
app.use(feedBackRoutes)
app.use(ordersRoutes)
app.use(usersRoutes)
conector.once('open', () => {
    socketIO.on('connection', (socket) => {
        socketIO_logic(socket)
    })
    streams_logic()
})
authSheets()
setTimeout(authSheets, 36000000)
```

Додаток Б. Приклад коду роутера

```
const express = require('express')
const {
    shortHistoryPost, createOrderPost, paymentPost,
    resultPaymentPost, changeOrderPost, getOrderByIdPost,
    ordersPost, addressPost, searchPost,
    createOrdersPost
} = require('../controllers/orders-controllers')
const router = express.Router()
router.post('/shortHistory', shortHistoryPost)
router.post('/createOrder', createOrderPost)
router.post('/payment', paymentPost)
router.post('/resultPayment', resultPaymentPost)
router.put('/userPage/changeOrder', changeOrderPost)
router.post('/userPage/getOrderByID', getOrderByIdPost)
router.post('/userPage/orders', ordersPost)
router.put('/userPage/saveAddress', addressPost)
router.post('/userPage/Admim/orderSearch', searchPost)
router.post('/userPage/Admin/createdOrders', createOrdersPost)
module.exports = router
```

Додаток В. Приклад коду контролера

```
const { BSON } = require('bson');
const PhoneCalls = require('../models/PhoneCalls')
const {phoneColection} = require('../server')
const callPost = (req, res) => {
    const { owner, name, phoneNumber, status } = req.body
    const result = new PhoneCalls({ owner, name, phoneNumber, status })
    result.save()
}
const deletePost = (req, res) => {
    const { id } = req.body
    const result = PhoneCalls.findByIdAndDelete({ _id: new BSON.ObjectId(id) }).then(res.send(id))
}
const adminCallPost = async (req, res) => {
    const result = await phoneColection.find().toArray().then(result => res.send(result))
}
module.exports = {
```

```
    callPost, deletePost, adminCallPost
}
```

Додаток Г. Приклад коду клієнтської частини

```
import { useContext, useEffect, useMemo, useRef, useState } from 'react'
import styles from './CreateOrder.module.css'
import Button from '../../../Button/Button'
import { CurrentContext } from '../../../Providers/CurrentProvider'
import axios from 'axios'
import {
    Combobox,
    ComboboxInput,
    ComboboxPopover,
    ComboboxList,
    ComboboxOption,
    ComboboxOptionText,
} from "@reach/combobox";
import "@reach/combobox/styles.css";
import { DirectionsRenderer, GoogleMap, Marker, Polyline, useLoadScript } from "@react-google-maps/api";
import usePlacesAutocomplete, { getGeocode, getLatLng } from 'use-places-autocomplete'
import { EmailContext } from '../../../Providers/EmailProvider'
import emailjs from '@emailjs/browser';
import { keyboard } from '../../../key'
import CheckOutForm from './CheckOutForm/CheckOutForm'
import { loadStripe } from "@stripe/stripe-js";
import { Elements } from "@stripe/react-stripe-js";
import { useNavigate } from 'react-router-dom'
const clearData = {
    name: '',
    phone: '',
    adress: {
        from: '',
        to: ''
    },
    item: '',
    weight: '',
    price: '',
    discount: '',
    status: 'created'
```

```
}

const clearData2 = {
    lat: '',
    lng: ''
}
let deleteCounter = 0
const stripePromise = loadStripe('');
export default function CreateOrder({ socket, isLoaded }) {
    const navigator = useNavigate()
    const { current, setCurrent } = useContext(CurrentContext)
    const { email, setEmail } = useContext(EmailContext)
    const [order, setOrder] = useState(clearData)
    const [error, setError] = useState(false)
    const [errorCalculator, setErrorCalculator] = useState(false)
    const [firstMarker, setFirstMarker] = useState(clearData2)
    const [secondMarker, setSecondMarker] = useState(clearData2)
    const [distanceF, setDistanceF] = useState()
    const [duration, setDuration] = useState()
    const [directionResponse, setDirectionResponse] = useState(null)
    const center = useMemo(() => ({ lat: 50.450001, lng: 30.523333 }), []);
    const [markers, setMarkers] = useState([])
    const [deleteMarker, setDeleteMarker] = useState(clearData2)
    const [save, setSave] = useState(false)
    const [clear, setClear] = useState(false)
    const [price, setPrice] = useState()
    const [prevPrice, setPrevPrice] = useState('')
    const [priceWithDiscount, setPriceWithDiscount] = useState('')
    const [discount, setDiscount] = useState({ discount: '' })
    const [payment, setPayment] = useState(false)
    const [choose, setChoose] = useState(false)
    const [paymentMethod, setPaymentMethod] = useState(1)
    const [checked, setChecked] = useState(1)
    const [paymentId, setPaymentId] = useState('')
    const [showForm, setShowForm] = useState(false)
    const [clientSecret, setClientSecret] = useState("");
    const [urlSucc, setUrlSucc] = useState(false)
    const [urlDen, setUrlDen] = useState(false)
```

```
const [handleId, setHandleId] = useState('')
const [finalResult, setFinalResult] = useState(false)
let routeOption = {
    directions: directionResponse,
}
const appearance = {
    theme: 'stripe',
};
const options = {
    clientSecret,
    appearance,
    handleId
};
let counter = 0
console.log(current)
console.log('handleId', handleId)



useEffect(() => {
    return () => {
        setFirstMarker(clearData2)
        setSecondMarker(clearData2)
        setDirectionResponse()
        setMarkers([])
    }
}, [])

function createOrder(e) {
    e.preventDefault()

    for (let element in order) {
        order[element] == '' ? element == 'discount' || element == 'price' ? '' : (console.log(element), counter++) : ''
    }
    console.log(counter)
    if (counter > 0) {
        setError(true)
    } else {
```

```
        priceWithDiscount > 0 ? (order.price = priceWithDiscount, order.discount = discount.discount) : (order.price =
price, order.discount = discount.discount)

        console.log('result -> ', order)

        axios.post('http://localhost:5000/createOrder', { ...order, owner: current.email, ownerName: current.name,
distance: distanceF, _id: current._id }).then(

            (res) => {

                console.log('Created Order ->', res.data)

                setPaymentId(res.data._id)

                if (current.info.address) {

                    setFinalResult(true)

                    setOrder(clearData)

                    setDeleteMarker(clearData2)

                    setFirstMarker(clearData2)

                    setSecondMarker(clearData2)

                    setError(false)

                    setSave(false)

                    setClear(true)

                } else {

                    setPayment(false)

                    setSave(true)

                }

                counter = 0

            })

        counter = 0

    }

}

useEffect(() => {

    async function showOnMap() {

        if (secondMarker.lat == '') return

        if (secondMarker.lat !== '') {

            const directionsService = new google.maps.DirectionsService()

            const results = await directionsService.route({

                origin: firstMarker,

                destination: secondMarker,

                travelMode: google.maps.TravelMode.DRIVING

            })

            console.log(results)

            setDirectionResponse(results)

            setDistanceF(results.routes[0].legs[0].distance.value)
```

```
            setDuration(results.routes[0].legs[0].duration.text)
        } else {
            setDirectionResponse(null)
            setDistanceF('')
            setDuration('')
            setFirstMarker(clearData2)
            setSecondMarker(clearData2)
            setPrice('')
            setPriceWithDiscount('')
        }
    }
    showOnMap()
}, [secondMarker, firstMarker])
console.log('from ->', order.adress.from)
console.log(directionResponse)
useEffect(() => {
    setMarkers(prev => [...prev, { ...firstMarker }])
}, [firstMarker])
useEffect(() => {
    setMarkers(prev => [...prev, { ...secondMarker }])
}, [secondMarker])
useEffect(() => {
    if (deleteMarker.lat == firstMarker.lat) {
        for (let element in markers) {
            console.log(markers[element])
            console.log(deleteMarker)
            if (markers[element].lat == deleteMarker.lat) {
                setMarkers(prev => prev.filter(item => item.lat !== markers[element].lat))
            }
        }
        setFirstMarker(clearData2)
        setDeleteMarker(clearData2)
    } else if (deleteMarker.lat == secondMarker.lat) {
        for (let element in markers) {
            if (markers[element].lat == deleteMarker.lat) {
                setMarkers(prev => prev.filter(item => item.lat !== markers[element].lat))
            }
        }
```

```
        setSecondMarker(clearData2)
        setDeleteMarker(clearData2)
    }
    setDirectionResponse(null)
    setDistanceF()
    setDuration()
    setPrice()
    setPriceWithDiscount()
    console.log('first marker -> ', firstMarker, '\nsecond marker -> ', secondMarker)
}, [deleteMarker])
console.log('markers ->', markers)
useEffect(() => {
    socket.on('updatedUserInfo', data => {
        console.log(data)
        setCurrent({ ...data })
    })
    return () => {
        socket.off('updatedUserInfo')
    }
}, [socket])
useEffect(() => {
    payment ? axios.post('http://localhost:5000/payment', { orderId: paymentId, _id: current._id, })
        .then(res => (console.log(res), setClientSecret(res.data.paymentIntent.client_secret),
            setHandleId(res.data.paymentIntent.description), setShowForm(true))) : ''
}, [payment])
console.log(directionResponse)
console.log('delete -> ', deleteMarker)
useEffect(() => {
    +order.weight
    order.weight == 0 || order.weight == '' ? setPrice(0)
        : order.weight <= 1 ? setPrice((distanceF / 1000) * 3)
            : order.weight >= 1 && order.weight < 3 ? setPrice((distanceF / 1000) * 5)
                : order.weight >= 3 && order.weight < 5 ? setPrice((distanceF / 1000) * 7)
                    : order.weight >= 5 && order.weight < 7 ? setPrice((distanceF / 1000) * 12)
                        : order.weight >= 7 && order.weight <= 10 ? setPrice((distanceF / 1000) * 15) : ''
    setDiscount({ discount: current?.discount })
    if (discount.discount > 0) {
        +price
```

```
        setPriceWithDiscount(price - (price * (discount.discount * 100) / 100))
    }
}, [price, discount.discount, order.weight, discount.discount, distanceF])
return (
    <div className={styles.items}>
        {
            error ? <div>
                <div className={styles.error}>
                    <h2>Створити замовлення</h2>
                    <div style={{
                        display: 'flex', flexDirection: 'column', alignItems: 'center', justifyContent: 'center', height: '400px'
                    }}>
                        <h2>Ви не заповнили усі поля </h2>
                        <Button margin={'15px 0px 0px 45px'} padding={0}
                            width={'400px'} borderRadius={'10px'} minHeightBtn={'80px'}
                            backgroundColor={'#fff200'} color={'black'}
                            display={'flex'} justifyContent={'center'} alignItems={'center'}
                            click={() => setError(false)}>OK</Button>
                    </div>
                </div>
            </div>
                :
                <div>
                    {
                        save ? <div className={styles.items}>
                            {
                                current?.info?.address == '' ? setSave(false)
                                    :
                                    <div style={{
                                        display: 'flex', flexDirection: 'column',
                                        justifyContent: 'center', alignItems: 'center'
                                    }}>
                                        <h2>Ви хочете зберегти введений адресс, як свій основний - {order.adress.from}</h2>
                                        <div style={{ display: 'flex' }}>
                                            <Button margin={'0px 10px 0px 0px'} padding={0}
                                                width={'125px'} borderRadius={'10px'} minHeightBtn={'60px'}
                                                backgroundColor={'#fff200'} color={'black'}
                                                display={'flex'} justifyContent={'center'} alignItems={'center'}
```

```
                                click={() => (axios.put('http://localhost:5000/userPage/saveAddress', { ...current,
newAdress: order.adress.from, socketId: socket.id }).then(
                                    setSave(false),
                                    setFinalResult(true),
                                    current.info.address = order.adress.from,
                                    setOrder(clearData),
                                    setDeleteMarker(clearData2),
                                    setFirstMarker(clearData2),
                                    setSecondMarker(clearData2),
                                    setError(false),
                                    setClear(true)))}
                            >Так</Button>
                            <Button margin={0} padding={0}
                                width={'125px'} borderRadius={'10px'} minHeightBtn={'60px'}
                                backgroundColor={'#fff200'} color={'black'}
                                display={'flex'} justifyContent={'center'} alignItems={'center'}
                                click={() => setSave(false)}
                            >Hi</Button>
                        </div>
                    </div>
                }
            </div>
            :
            <div>
                {
                    showForm ?
                        <div style={{
                            display: 'flex', flexDirection: 'column', alignItems: 'center',
                            justifyContent: 'center'
                        }}>

                            <h2>Оплата</h2>

                            <div style={{
                                display: 'flex', justifyContent: 'center', padding: '0px 0px 0px 50px'
                            }}>
                                {clientSecret && (
                                    <Elements options={options} stripe={stripePromise}>
```

```
                                    <CheckOutForm setUrlSucc={setUrlSucc} setUrlDen={setUrlDen}
handleId={handleId} paymentId={paymentId} />
                                </Elements>
                            )}
                        </div>
                    </div>
                    :
                    <div>
                        {
                            finalResult ? <div style={{
                                display: 'flex', flexDirection: 'column', alignItems: 'center'
                            }}>
                                <h2>Оплата</h2>
                                <h4 style={{
                                    fontSize: '20px', color: 'red', marginBottom: '20px'
                                }}>*Після оплати ваше замовлення автоматично перейд в опрацювання</h4>
                                <div style={{
                                    display: 'flex'
                                }}>
                                    <Button click={() =>
navigator(`/userPage/User/order/unsuccsessful/${paymentId}`)}
                                        margin={'0px 50px 0px 0px'} padding={0}
                                        width={'300px'} borderRadius={'10px'} minHeightBtn={'80px'}
                                        backgroundColor={'#fff200'} color={'black'}
                                        display={'flex'} justifyContent={'center'} alignItems={'center'}>
                                        Перейти на замовлення</Button>
                                    <Button click={() => { setPayment(true) }}
                                        margin={0} padding={0}
                                        width={'300px'} borderRadius={'10px'} minHeightBtn={'80px'}
                                        backgroundColor={'#fff200'} color={'black'}
                                        display={'flex'} justifyContent={'center'} alignItems={'center'}>
                                        Оплатити</Button>
                                </div>
                            </div>
                                :
                                <div style={{
                                    display: 'flex', flexDirection: 'column', alignItems: 'center',
                                    justifyContent: 'center'
                                }}>
```

```
<h2>Створити замовлення</h2>

<div className={styles.itemContainer}>

  <div className={styles.item}>

    <div>
      <p>Як вас звати:</p>
      <input onChange={e => setOrder(prev => ({
        ...prev, name: e.target.value
      }))} value={order.name} placeholder="Імя" />
    </div>
    <div>
      <p>Вкажіть свій телефон:</p>
      <input onChange={e => setOrder(prev => ({
        ...prev, phone: e.target.value
      }))} value={order.phone} placeholder="Телефон" onKeyDown={(e) => {
        if (keyboard.includes(e.keyCode) || keyboard.includes(e.key)) {
          console.log('ok')
          e.preventDefault()
        }
      }} />
    </div>

    <div style={{
      display: 'flex'
    }}>
      <div>
        <p>Вкажіть маршрут перевезення:</p>
        <MarkerInput clear={clear} setClear={setClear}
currentAddress={current?.info?.address} order={order} setMarker={setFirstMarker}
setDeleteMarker={setDeleteMarker} setDistanceF={setDistanceF} first={true} />
        <MarkerInput clear={clear} setClear={setClear} order={order}
setMarker={setSecondMarker} setDeleteMarker={setDeleteMarker} setDistanceF={setDistanceF} first={false} />
      </div>

      <div>
        <div>
          <p style={{
```

```
                marginTop: '40px',
                marginBottom: '10px'
            }}>Дистанція:</p>
            <span>{distanceF ? distanceF / 1000 : 0} км</span>
        </div>

        <div style={{
            marginTop: '40px'
        }}>
            <p style={{ margin: '0px 0px 10px 0px' }}>Ціна:</p>
            <span>{price ? ~~price : 0} грн</span>
        </div>

        <div style={{
            marginTop: '40px'
        }}>
            <p style={{ margin: '0px 0px 10px 0px' }}>Ціна якщо є
знижка:</p>
            <span>{priceWithDiscount ? ~~priceWithDiscount : '-'}</span>
        </div>
    </div>
</div>

<div>
    <p>найменування вантажу:</p>
    <input onChange={e => setOrder(prev => ({
        ...prev, item: e.target.value
    }))} value={order.item} placeholder="Вантаж" />
</div>

<div>
    <p>Вага вантажу (<span style={{
        fontWeight: 500
    }}>в тонах</span>):</p>
    <input onChange={e => setOrder(prev => ({
        ...prev, weight: e.target.value
    }))} value={order.weight} placeholder="Вага" onKeyDown={(e) => {
        if (keyboard.includes(e.keyCode) || keyboard.includes(e.key)) {
```

```
                    console.log('ok')
                    e.preventDefault()
                }
            }} />
        </div>

        <div>
            <Button click={createOrder}
                margin={0} padding={0}
                width={'400px'} borderRadius={'10px'} minHeightBtn={'80px'}
                backgroundColor={'#fff200'} color={'black'}
                display={'flex'} justifyContent={'center'} alignItems={'center'}>
                Відправити заявку</Button>
        </div>
    </div>

    <div style={{
        display: 'flex', alignItems: 'center'
    }}>
        {!isLoaded ? (
            <h1>Loading...</h1>
        ) : (
            <GoogleMap
                mapContainerClassName={styles.mapContainer}
                center={firstMarker.lat !== '' ? firstMarker : secondMarker.lat !== '' ?
secondMarker : center}
                zoom={12}
            >
                {
                    markers.map(item => (
                        item.lat == '' ? <Marker visible={false} /> :
                            <Marker position={item} />
                    ))
                }
                {
                    secondMarker.lat == '' || firstMarker.lat == '' ? '' : <DirectionsRenderer
options={routeOption} />
                }
            </GoogleMap>
```

```
                    )}
                  </div>
                </div>
              </div>
            }
          </div>
        }
      </div>
```